KU LEUVEN

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN

CAMPUS DE NAYER

Project Web-AI

15/01/2024

Aaron Van Campenhout

Academiejaar 2023-2024

1 Knowledge Graph Construction

1.1 Creating ontology in Protégé

I started creating a basic ontology with Protégé, I focused on outlining the classes, object properties, and data properties I intended to include.

To get ideas and inspiration, I explored various ontologies related to music available at https://lov.linkeddata.es/dataset/lov/terms?q=music. However, I encountered a mismatch between these existing ontologies and the datasets I'm working with:

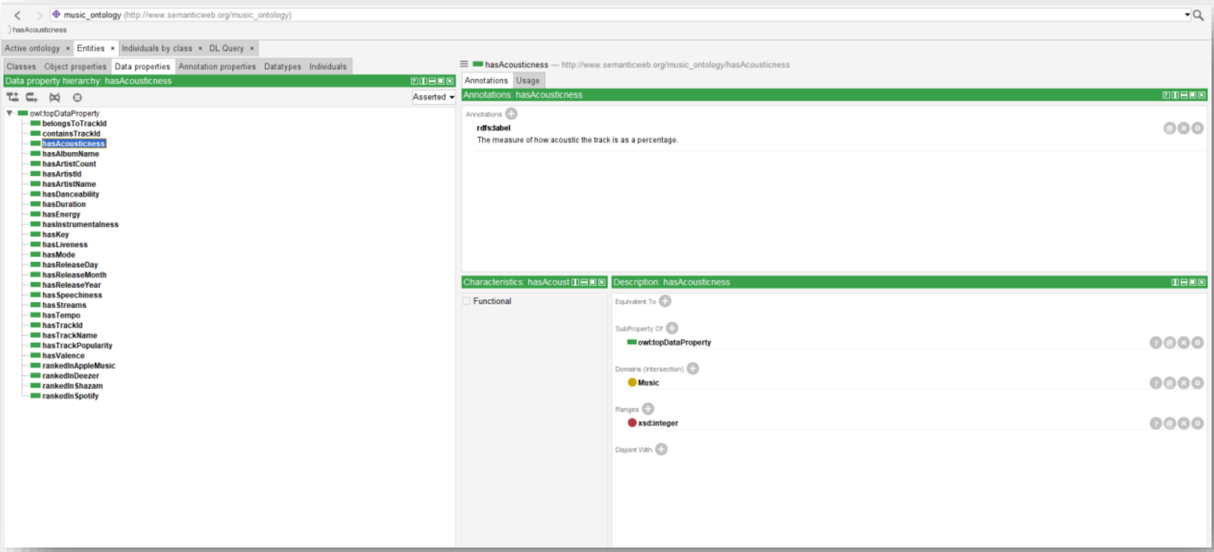
- https://www.kaggle.com/datasets/sveta151/tiktok-popular-songs-2022/data
- https://www.kaggle.com/datasets/nelgiriyewithana/top-spotify-songs-2023

I first created my classes:

1. **Track**
2. **Artist**
3. **Music**
4. **Chart**
5. **Playlist**

I decided to first define the data properties by looking at the columns of my two datasets. I noticed that some columns didn't align correctly with their designated names and corresponding values, but this issue I addressed in the preprocessing step of my data, which I will talk about later.

I wrote down the columns I intended to merge from both datasets and those I opted to maintain unchanged. Ultimately, I chose to incorporate all the data properties that were specified in the image below.


music_ontology (http://www.semanticweb.org/music_ontology)
hasAcousticness
Active ontology
Entities
Individuals by class
DL Query
Classes
Object properties
Data properties
Annotation properties
Datatypes
Individuals
Data property hierarchy: hasAcousticness
Asserted
hasAcousticness — http://www.semanticweb.org/music_ontology/hasAcousticness
Annotations
Usage
Annotations: hasAcousticness
Annotations
Characteristics: hasAcoust
Description: hasAcousticness
Equivalent To
SubProperty Of
Domains (intersection)
Ranges
Disjoint With

- owl:topDataProperty
  - belongsToTrackId
  - containsTrackId
  - hasAcousticness
  - hasAlbumName
  - hasArtistCount
  - hasArtistId
  - hasArtistName
  - hasDanceability
  - hasDuration
  - hasEnergy
  - hasInstrumentalness
  - hasKey
  - hasLiveness
  - hasMode
  - hasReleaseDay
  - hasReleaseMonth
  - hasReleaseYear
  - hasSpeechiness
  - hasStreams
  - hasTempo
  - hasTrackId
  - hasTrackName
  - hasTrackPopularity
  - hasValence
  - rankedInAppleMusic
  - rankedInDeezer
  - rankedInShazam
  - rankedInSpotify

rdfs:label
The measure of how acoustic the track is as a percentage.

Functional

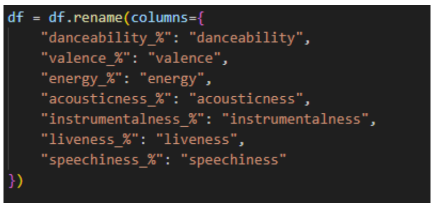
SubProperty Of: owl:topDataProperty
Domains (intersection): Music
Ranges: xsd:integer

Finally, I created all the object properties:

1. **performedBy**: domain is track and range is artist.
2. **hasMusicDetails**: domain is track and range is music.
3. **chartContains**: domain is chart and range is track.
4. **playlistContains**: domain is playlist and range is track.

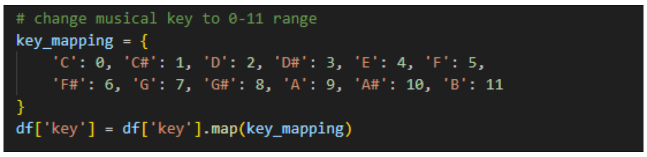
1.2 Preprocessing data

As referenced in section 1.1, there was a need to merge certain columns and harmonize their content. To achieve this, I initially extracted the data from both CSV files into a Pandas dataframe. Following this, I commenced the process of renaming specific columns to address minor discrepancies between them.



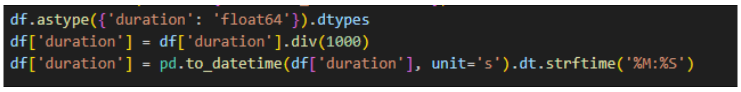
```
df = df.rename(columns={
    "danceability_%": "danceability",
    "valence_%": "valence",
    "energy_%": "energy",
    "acousticness_%": "acousticness",
    "instrumentalness_%": "instrumentalness",
    "liveness_%": "liveness",
    "speechiness_%": "speechiness"
})
```

Then I noticed that the musical key in the first datasets was declared as a string and in the second one as an integer. I chose to use the integer values:



```
# change musical key to 0-11 range
key_mapping = {
    'C': 0, 'C#': 1, 'D': 2, 'D#': 3, 'E': 4, 'F': 5,
    'F#': 6, 'G': 7, 'G#': 8, 'A': 9, 'A#': 10, 'B': 11
}
df['key'] = df['key'].map(key_mapping)
```

I changed all the properties regarding the music in the second dataset. There they used as value a float datatype, ranging from 0.0 to 1.0, to define these characteristics: acousticness, energy, danceability… The duration I changed as well, so I could use xsd:DateTime to represent it’s value.



```
df.astype({'duration': 'float64'}).dtypes
df['duration'] = df['duration'].div(1000)
df['duration'] = pd.to_datetime(df['duration'], unit='s').dt.strftime('%M:%S')
```

Next I combined both dataframes, this would result in some NA values, which I filled in afterwards with some default values.

After this I wrote a few functions for creating some intermediate tables so I could connect the different classes with each other using the defined object properties.

I ended up with five different csv files, containing all the data necessary to create the data and object properties.

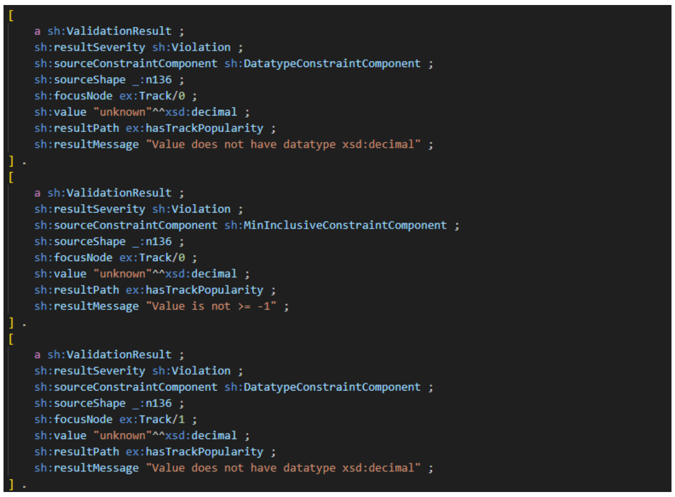
1.3 Writing mapping rules in RML

To start, I wrote the mapping rules in YARRRML, which were later transformed into RML format by Matey. This RML file served as the foundation for generating a Knowledge Graph.

However, I encountered challenges when debugging errors within my mapping rules as Matey didn't provide explanations for the issues. Next time, I would suggest to utilize yatter for RML generation and rely on RMLMapper for producing RDF triples (Knowledge Graph) due to their efficiency and error response capabilities.

2 KNOWLEDGE GRAPH VALIDATION

While executing the SHACL rules I had written, I encountered issues with certain values. These problems stemmed from an error made during the preprocessing stage when merging both datasets. Specifically, filling empty fields with "unknown" posed a challenge as some of these fields were originally defined in the RML rules as a “xsd:integer”. Consequently, this mismatch caused a multitude of errors in the validation report, specified in the image below.

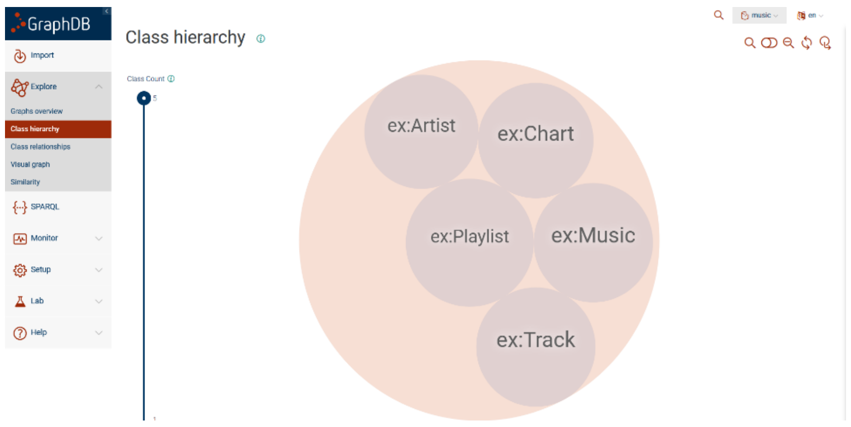


```
[
    a sh:ValidationResult ;
    sh:resultSeverity sh:Violation ;
    sh:sourceConstraintComponent sh:DatatypeConstraintComponent ;
    sh:sourceShape _:n136 ;
    sh:focusNode ex:Track/0 ;
    sh:value "unknown"^^xsd:decimal ;
    sh:resultPath ex:hasTrackPopularity ;
    sh:resultMessage "Value does not have datatype xsd:decimal" ;
] .
[
    a sh:ValidationResult ;
    sh:resultSeverity sh:Violation ;
    sh:sourceConstraintComponent sh:MinInclusiveConstraintComponent ;
    sh:sourceShape _:n136 ;
    sh:focusNode ex:Track/0 ;
    sh:value "unknown"^^xsd:decimal ;
    sh:resultPath ex:hasTrackPopularity ;
    sh:resultMessage "Value is not >= -1" ;
] .
[
    a sh:ValidationResult ;
    sh:resultSeverity sh:Violation ;
    sh:sourceConstraintComponent sh:DatatypeConstraintComponent ;
    sh:sourceShape _:n136 ;
    sh:focusNode ex:Track/1 ;
    sh:value "unknown"^^xsd:decimal ;
    sh:resultPath ex:hasTrackPopularity ;
    sh:resultMessage "Value does not have datatype xsd:decimal" ;
] .
```

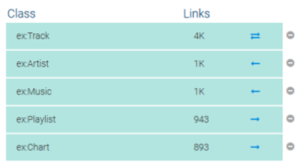
Returning to the preprocessing stage, I corrected the values according to their respective data types. This adjustment resolved the issue and ensured there were no errors in the validation report. With everything in order, I proceeded to load my “graph.ttl” file into my SPARQL endpoint.

3 KNOWLEDGE GRAPH CONSUMPTION

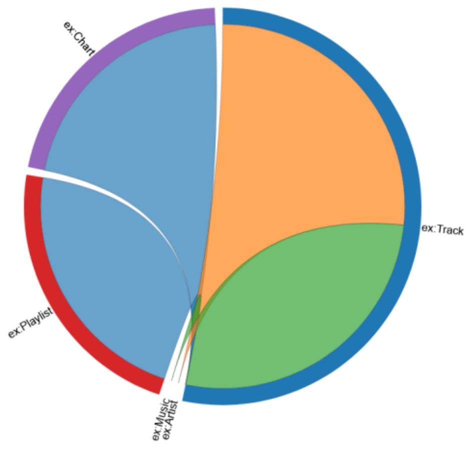
While importing the data into GraphDB, I encountered initial difficulties with the relationships between classes. Upon reviewing the “music.yaml” mapping rules, I discovered some typos that were causing these issues. After several attempts and corrections, I successfully established all the necessary classes within the database.



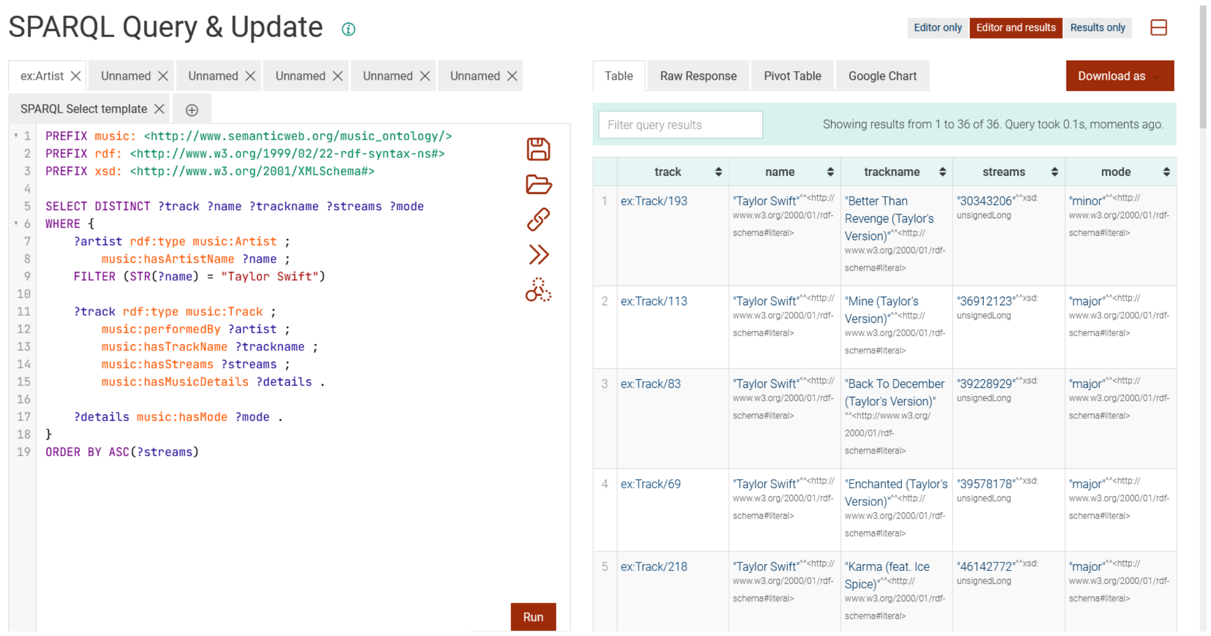
The visualization of connections between classes in the Explore section was particularly intriguing to observe.



Class	Links	
ex:Track	4K	⇄
ex:Artist	1K	←
ex:Music	1K	←
ex:Playlist	943	→
ex:Chart	893	→



After that, I wrote a few interesting queries to check if all the data had been loaded correctly. The next query finds songs from a specific artist, and will order it by ascending streams.


SPARQL Query & Update

```
PREFIX music: <http://www.semanticweb.org/music_ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?track ?name ?trackname ?streams ?mode
WHERE {
    ?artist rdf:type music:Artist ;
        music:hasArtistName ?name ;
    FILTER (STR(?name) = "Taylor Swift")

    ?track rdf:type music:Track ;
        music:performedBy ?artist ;
        music:hasTrackName ?trackname ;
        music:hasStreams ?streams ;
        music:hasMusicDetails ?details .

    ?details music:hasMode ?mode .
}
ORDER BY ASC(?streams)
```

Showing results from 1 to 36 of 36. Query took 0.1s, moments ago.

	track	name	trackname	streams	mode
1	ex:Track/193	"Taylor Swift"^^<http://www.w3.org/2000/01/rdf-schema#literal>	"Better Than Revenge (Taylor's Version)"^^<http://www.w3.org/2000/01/rdf-schema#literal>	"30343206"^^xsd:unsignedLong	"minor"^^<http://www.w3.org/2000/01/rdf-schema#literal>
2	ex:Track/113	"Taylor Swift"^^<http://www.w3.org/2000/01/rdf-schema#literal>	"Mine (Taylor's Version)"^^<http://www.w3.org/2000/01/rdf-schema#literal>	"36912123"^^xsd:unsignedLong	"major"^^<http://www.w3.org/2000/01/rdf-schema#literal>
3	ex:Track/83	"Taylor Swift"^^<http://www.w3.org/2000/01/rdf-schema#literal>	"Back To December (Taylor's Version)"^^<http://www.w3.org/2000/01/rdf-schema#literal>	"39228929"^^xsd:unsignedLong	"major"^^<http://www.w3.org/2000/01/rdf-schema#literal>
4	ex:Track/69	"Taylor Swift"^^<http://www.w3.org/2000/01/rdf-schema#literal>	"Enchanted (Taylor's Version)"^^<http://www.w3.org/2000/01/rdf-schema#literal>	"39578178"^^xsd:unsignedLong	"major"^^<http://www.w3.org/2000/01/rdf-schema#literal>
5	ex:Track/218	"Taylor Swift"^^<http://www.w3.org/2000/01/rdf-schema#literal>	"Karma (feat. Ice Spice)"^^<http://www.w3.org/2000/01/rdf-schema#literal>	"46142772"^^xsd:unsignedLong	"major"^^<http://www.w3.org/2000/01/rdf-schema#literal>

While doing some queries, I realized that the relationships were unidirectional. To address this, I included a unique ID in every class, enabling me to do searches without solely relying on relationships.

You can find more queries in the following file "query.md", later in the text I provide a link to my github repository.

4 LESSONS LEARNED

In my recent work, I've found two important ways to make data handling smoother and more accurate.

First, I learned that dealing with missing values at the end of the data preprocessing stage works better. It helps keeping things organized and reduces errors when doing validation, by giving a clearer picture of what's missing.

Second, I discovered that using RMLMapper early on when creating RML mappings is a big help. It gives us more information about errors, making it easier to understand and fix issues in the mappings.

These practices could have improved my workflow, making data preprocessing more efficient and cutting down on errors when creating RML mappings.

5 Future steps

The artist class I developed currently comprises only the artist's name and ID without any additional properties. However, if a dataset containing more information about artists emerges in the future, this class can be expanded by incorporating additional data properties.

6 Results

All my results you can find on my GitHub.

https://github.com/aarontenzing/Web-AI

All the csv files:

- tracks.csv: contains the combined datasets with all the information
- artists.csv
- charts.csv
- playlists.csv
- music.csv

Preprocessing program:

- preprocess_data.py

Directory with all the deliverables:

- r0843851-ontology.tll
- r0843851-mapping.rml
- r0843851-graph.tll
- r0843851-shapes.tll
- r0843851-report.tll
- r0843851-queries.txt

Four interesting queries:

- query.md